Getting Acquainted with Unix and C: Approximating π

Lucais Sanderson

22 January 2023

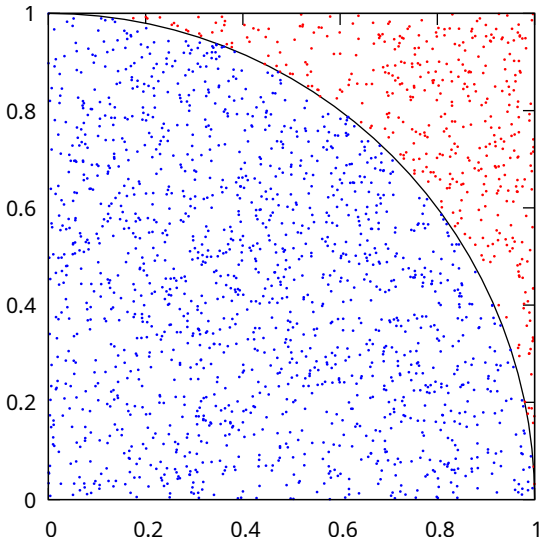
1 Intro

At the top level, running the shell script, `plot.sh`, takes data generated from `monte_carlo` and generates several graphs in PDF form in the local directory. `plot.sh` accomplishes this by utilizing `gnuplot` to plot the respective graphs.

2 Circle Graph

The first graph the `plot.sh` script generates is a quarter circle with a few thousand points lying within and out of the circle. These points are puled from the output that `monte_carlo` generates. This is due to how `monte_carlo` approximates π. `monte_carlo` uses a random seed to generate a given number of points in the domain/range $x : [0, 1], y : [0, 1]$. If the point lies in the unit circle, then the point is marked as such. This collection of points are used to calculate the approximation. The graph takes all of the generated points, plots them and if the point is in the circle, it's colored blue, otherwise it's colored red.

The table below shows the output from `monte_carlo` which is used to create the respective graph below it.



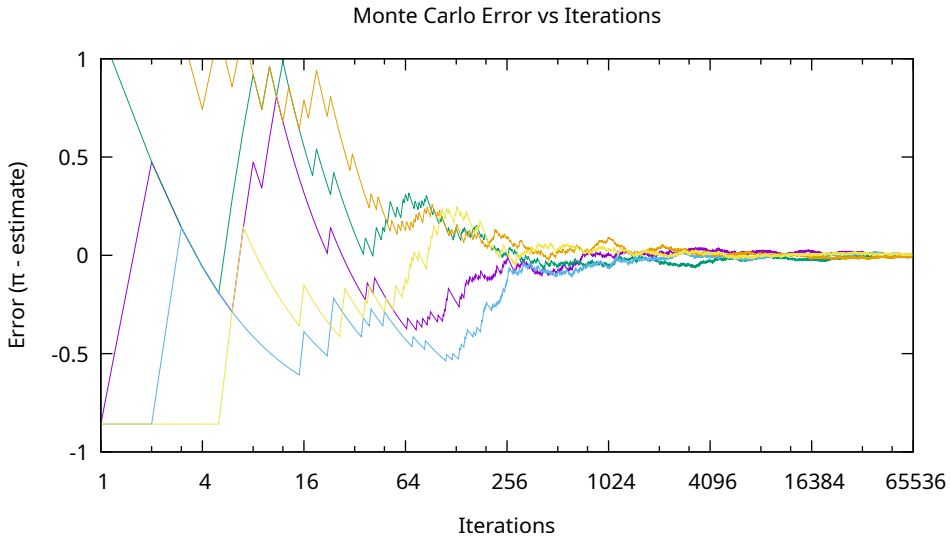
3 Error Graph

The error graph plots the Error vs Iteration derived from `monte_carlo` output. "Error" is calculation by

$$3.14159265358979323846264338327950288419 7 - calculated\_value$$

which hones in on 0 as the number of iterations get very high. The script loops the `monte_carlo` program, waiting for a new seed to be generated (seed based on time), so that several different lines are plotted. The x axis is log-scaled so the graph is more readable.

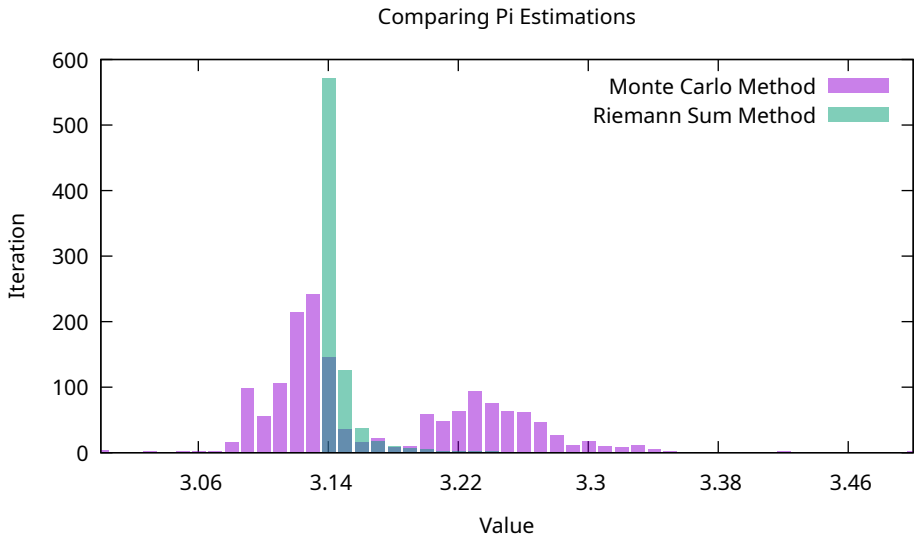
The graph below shows what a random set of seeds, each generated one second apart, would create on a Error vs Iterations graph.



4 Comparison Graph

This graph plots the result from the Monte Carlo method versus a different method using calculus: Riemann sum method. This is using only 800 iterations for both methods. This isn't nearly enough for Monte Carlo to be significantly accurate but that shows how effective the Riemann method is.

Below is an example of the output generated.



5 Wrap-Up

Closing out of this assignment, I've learned a *lot* about scripting, graphing using `gnuplot`, and the basics of C. Additionally I learned a little bit about Latex, which I'm using right now.
Content-wise, I come away from this assignment understanding 2 methods of approximating π, a rudimentary comparison of their effectiveness, and how to represent the data using these methods.

Appendices

List of UNIX and `gnuplot` commands used:

- `make all`
  - Runs Makefile in the current directory.
- `rm r.dat mc.dat`
  - Remove temp data files used for `gnuplot` commands.
- `gnuplot < x.plot`
  - Redirects `x.plot` to `gnuplot` to run the sequence of commands in `x.plot`. Where x is the file name and `.plot` is the extension.

- `plot for [i = 0:4] "< (./monte_carlo -n 20000 | awk '{print $1, 3.1415926... - $2 }' | grep -v 'I') sleep 1" with lines title ""`
  - Runs the `monte_carlo` binary, pipes the out to `awk` to extract necessary fields. Data then `grep`'d to remove header row. All this being in a command line sequence I then used sleep to give time for the seed to change in `monte_carlo`. All this is redirected into `plot`. I found the `grep -v` command from user codaddict at url: stackoverflow.com
- `set logscale x 4`
  - Sets the x axis of the plot to a logarithmic scale.
- `set object circle, set size ratio -1`
  - Creates circle object to be in the plot. Derived "set object circle ..." (line) and "set size ratio -1" from users anyras and mgilson respectively at URL: stackoverflow.com
- `set palette defined`
  - Will associate color to line or dot of plot based on a field value.